The 4th International Workshop on Software Engineering for HPC in Computational Science and Engineering

The 4th International Workshop on Software Engineering for High Performance Computing in Computational Science & Engineering

Abstract

Despite the increasing demand for utilizing HPC for CSE applications, software development for HPC historically attracted little attention from the SE community. Paradoxically, the HPC CSE community has increasingly been adopting SE techniques and tools. Indeed, the development of CSE software for HPC differs significantly from the development of more traditional business inforåmation systems, from which many SE best practices and tools have been drawn.

This workshop, the fourth in the series to be co-located with the Supercomputing conference series, examined two main topics: testing and trade-offs. Through presentations of work in this area and structured group discussions, the participants highlighted some of the key issues, as well as indicating the direction the community needed to go. In particular, there is a need for more high-quality research in this area that we can use as an evidence base to help developers of CSE applications to change practice and benefit from advances in software engineering.

Index Terms

Software Engineering, Computational Science, Computational Engineering, High Performance Computing

Introduction

Within the Computational Science & Engineering (CSE) community, there are few places to publish research related to the unique SE challenges faced by CSE developers along with the approaches identified to address those challenges. Within the HPC community, this is constrained further. The goal of this workshop is to provide a unique venue for researchers from SE and CSE working with HPC applications and tools to interact and discuss issues relevant to the intersection of their fields. By bringing these groups together, our goal is to support the building of a common research agenda to deal with the complex software development issues present in the CSE on HPC domain. Furthermore, the discussion among these two groups of researchers will be invaluable in identifying those aspects of SE that should be considered for HPC education programs. New researchers are coming into

this line of research and are often unaware of each other's work. There is no one preferred journal for publication or other readily found source for researchers with this common interest, therefore this workshop is an important focal point.

This workshop is one in a series of workshops examining the interaction between Software Engineering and Scientific (Engineering) Software. The workshop website (http://www.SE4Science.org/workshops) has links to current, previous, and upcoming workshops in the series. The workshop summarized in this article was co-located with Supercomputing 2016. It was concerned with identifying and applying appropriate software engineering (SE) tools and practices (e.g., code generators, static analyzers, validation + verification (V&V) practices, design approaches, and maintenance practices) to support and ease the development of Computational Science & Engineering (CSE) software for High Performance Computing (HPC). Specifically:

- CSE applications, which include, large parallel models/simulations of the physical world running on HPC systems.
- CSE applications that utilize HPC systems (e.g., GPUs computing, compute clusters, or supercomputers) to manage and/or manipulate large amounts of data.

Despite the increasing demand for utilizing HPC for CSE applications, software development for HPC has historically attracted little attention from the SE community. Paradoxically, the HPC CSE community has increasingly been adopting SE techniques and tools. Indeed, the development of CSE software for HPC differs significantly from the development of more traditional business information systems, from which many SE best practices and tools have been drawn. These differences appear at various phases of the software lifecycle:

- Requirements (e.g., constantly changing, risky due to exploration of the unknown)
- Design (e.g., complex communication, parallelism, and fault tolerance)
- V&V (e.g., expected results are often unknown, existing tools must be adapted)
- Deployment (e.g., dealing with long project lifespans)

Therefore, in order to identify and develop appropriate tools and practices to support HPC CSE software, members of the SE community, the CSE community, and the HPC community must interact with each other. This workshop provides a platform to facilitate this interaction by encourage paper submission and workshop participation by people from all three communities. In addition to presentation and discussion of the accepted papers, significant time during the workshop was devoted to large and small group discussions among the participants to identify important research questions at the intersection of SE and HPC CSE that are in need of additional study.

Previous editions of this workshop [9-14] have focused discussion around a number of interesting topics, including: bit-by-bit vs. scientific validation, reproducibility, unique characteristics of CSE software that affect software development choices, major software quality goals for CSE software, crossing the communication chasm between SE and CSE, measuring the impact of SE on scientific productivity, SE tools and methods needed by the CSE community, and how to effectively test CSE software.

Motivated by the discussion during the 2015 edition of this workshop, we expanded the previous workshops by adding two special focus areas. First, we placed special emphasis on

experience reports (including positive, negative, and neutral) of applying software engineering practices to the development of HPC scientific software. It is important to document those successes and failures for the community. Second, as quality assurance is a challenge in the scientific HPC domain, we also recruited papers describing quality assurance techniques for HPC science and their use in practice.

Summary of the workshop

Based on the accepted papers (http://conferences.computer.org/sehpccse/2016/), we organized the workshop into two thematic sessions. Each session began with the presentation of accepted full papers and extended abstracts. Full papers reported on mature research with the aim of increasing the overall level of knowledge amongst the workshop participants, and making them consider the direction of the field. Extended abstracts described early stage research with the aim of sparking interesting discussion among the workshop participants.

After each set of presentations, we asked the workshop participants to break-up into smaller groups to discuss ideas related to the theme. The group discussions are the most important aspect of the workshop because they allow members of the different communities to interact on a more direct basis, and understand eachothers' perspectives. In total, over 45 people participated in the workshop, from at least three continents. The following sections describe the papers and discussion from each session.

Session 1: Testing

This session contained three full papers and one short papers, summarized as follows. First, *Single-Sided Statistic Multiplexed High Performance Computing* by Justin Shi and Yasin Celik (full paper) presented research evaluating the performance of two implementations of single-sided statistic multiplexed computing (SMC). The results show that single-sided SMC is a promising direction for future extreme scale HPC but the work also explored some of the issues of testing HPC applications [1]. Second, *A Case Study: Test-Driven Development in a Microscopy Image-Processing Project* by Aziz Nanthaamornphong (full paper) used a qualitative case study with automatically gathered data along and developers surveys to investigate the use of Test-Driven Development (TDD) for scientific HPC software development. The results showed that TDD was difficult to use in a parallel computing environment. Specifically, developers found (1) writing tests was difficult, (2) TDD is time consuming, and (3) the lack of general software engineering knowledge was a hindrance [2]. Third, *Towards Automatic and Flexible Unit Test Generation for Legacy HPC Code* by Christian Hovy and Julian Kunkel (full paper) addresses the difficulty developers of HPC applications face when trying to write unit tests by providing a "capture and replay" approach that extracts data from the running application to be used as test input data. The solution provides a code generator to create the basic test driver and test data, which can be extended by the developer [3]. Fourth, *Towards an Empirical Study Design for Concurrent Software Testing* by Silvana M. Melo, Paulo S. L. Souza and Simone R. S. Souza (extended abstract) describes the findings from a literature review about concurrent software testing

techniques. These observations serve as the basis for the proposal of an empirical study design to compare various concurrent software testing approaches [4].

The questions that arose during this first set of presentations included:

- Does modern hardware make it harder to study the effects of applying software engineering techniques to HPC?
- Is there enough evidence to support Test Driven Design for HPC?
- What additional research would aid the community's understanding of concurrent software testing?

The first topic to be considered, following on from the first session on testing, was: **what is best practice for testing for HPC applications?**

The workshop raised a number of key issues which made it hard to move forward with specific, technical recommendations. Firstly, the stakes in scientific software are often not seen as being high enough to make testing important ("No one dies from a bug in the code") despite the fact that in many cases, scientific software is literally used in applications that will cause people to live or die, such as calculation of radiotherapy treatment, or simulation of the effects of climate change on an area. Secondly, scientists typically want to see overwhelming evidence before they change practice, but we as a community can not point to good papers that show measurable benefits. Compounding this, many of the better papers that show a significant benefit are based on data collected by studying undergraduate coding projects which are not large enough to be perceived as relevant or in realistic conditions: researchers want data on productivity for relevant types of software, with relevant types of developers. Thirdly, the issue of testing for numerical correctness - how do you know it is still producing the right results? - was brought up again (it has been a feature of previous workshops). This however led on to the agreement that different domains are likely to have different testing needs, and that maybe there are different "qualities of correct solutions" that can be identified. Finally, it was noted that spending time on validation testing is important, but that developers shouldn't forget verification testing.

The participants also considered the practicalities of improving testing practice. Hiring a dedicated test engineer was an obvious solution but not within every project's reach or budget. It was however noted that pair programming with a scientist and a software engineer has been successful. There were many specific recommendations for improving test practice for developers / projects, including: do not write obfuscating code; build and maintain a good testing infrastructure; use existing testing frameworks rather than writing your own (do not reinvent the wheel), and, encourage development of small procedures with short interfaces as it makes testing much more feasible. The discussion on the talk by Christian Hovy on automatic unit test generation for legacy HPC code suggested that the capture and replay strategy discussed in his presentation will be very valuable to the HPC community, augmenting existing regression testing by supporting creation of fine-grained regression tests.

Finally, it was noted that it would be impossible to teach any particular group or community about the whole panoply of software engineering and hope to produce sustained change.

There are many topics that are relevant; the key is to pick one change of practice and get that right first. There is also a case for teaching the basics of best practices in software development in terms of hardware architecture in the HPC community. Existing discussions in this area were noted, including by Software Carpentry (http://software-carpentry.org/blog/2014/10/why-we-dont-teach-testing.html); and Close Enough for Scientific Work (https://github.com/swcarpentry/close-enough-for-scientific-work).

Session 2: Trade-offs

This session contained three full papers and one short papers, summarized as follows. First, *Advantages, Disadvantages and Misunderstandings About Document Driven Design for Scientific Software* by Spencer Smith, Thulasi Jegatheesan, and Diane Kelly (full paper) describes a study of the use of a document-driven design process on five scientific computing software projects. The code owners agreed that a systematic development process can be beneficial. While they had positive or neutral responses to the redeveloped artifacts, there was some concern about the time required [5]. Second, *The Scalability-Efficiency/Maintainability-Portability Trade-Off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review* by Dirk Pflüger and colleagues (full paper), describes a literature review to summarize proposed solutions for the trade-off and discuss findings from simulation software engineering. The overall observation from the work is that there is not yet a strong empirical foundation upon which to draw conclusions and additional study is needed [6]. Third, *Computational Efficiency vs. Maintainability and Portability. Experiences with the Sparse Grid Code SG++* by Dirk Pflüger and David Pfander demonstrates some optimizations applied to a numerics framework for sparse grids. The paper describes the trade-off between the performance advantages and the software quality disadvantages [7]. Fourth, *Code Complexity versus Performance for GPU-accelerated Scientific Applications* by A WK. Umayanganie Munipala and Shirley Moore (extended abstract) compares the performance of CUDA, OpenCL, and OpenACC on three benchmark codes using GPUs. The results show that CUDA and OpenCL have significantly more lines of code than OpenACC and that CUDA and OpenCL have slightly better performance that optimized OpenACC [8].

The questions that arose during the second set of presentations included:

- Does the fact that many studies used undergraduate students as proxies for the actual professionals undertaking the software development affect the outcomes of the research and any recommendations based on them?
- Do we have an understanding of whether we expect results to change when dealing with cross-disciplinary / inter-disciplinary fields?
- How can we improve the efficiency of code generation by avoiding code duplication?

The second topic to be considered, following on from the second session on trade-offs, was: **what are the key trade-offs in developing HPC applications and how does this affect software engineering practice?**

The workshop participants noted that HPC software faces a large number of trade-offs related to maintainability, portability, scalability - understanding which trade-off is the most

important for you can help you find the solution. As an example, straightforward code and modular code are a preference in some contexts. However, whilst highly optimised code may be hard to understand, it is sometimes necessary to get results in a reasonable time. A key point is longevity and maintainability of the code. Hiding complexity may come back and bite you, especially when the architecture changes and the person who wrote the underpinning code leaves. Often, the trade-off is between the developer's time and the machine's time (man-hours vs core-hours). However build and installation difficulties restrict portability and choices.

Another significant trade-off is between software quality and academic credit. Because the current incentives promote short-term results over long-term reuse, it can be hard to instill better development models and practices that may have (perceived or real) up-front costs. Finally, there was much discussion about the trade-offs that were made in research studies in this area. These included variances in studies, e.g. how does the performance of a language depend on the person writing it, and what studies we still needed to conduct, e.g. performance vs portability and financial cost vs other factors.

Conclusions

This workshop achieved its goal of providing a venue for researchers from SE and CSE working with HPC applications and tools to interact and discuss issues relevant to the intersection of their fields. However it is clear that a number of issues have been raised that prevent the work being done by these researchers being adopted by the wider CSE community. The foremost of these is that there is a lack of high-quality research in this area that we can show to developers of scientific software - we lack the evidence base that will persuade them to change practice. Therefore there is clearly a role for this workshop series to enable these studies to take place, and we hope that we will see the outcomes presented in future workshops. We also hope that this CiSE column will provide a venue for publishing the types of high-quality research results that are needed to change practice.

References

1. Shi, J. and Celik, Y., "Single-Sided Statistic Multiplexed High Performance Computing," *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.9
2. Nanthaamornphong, A., "A Case Study: Test-Driven Development in a Microscopy Image-Processing Project." *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.5
3. Hovy, C. and Kunkel, J., "Towards Automatic and Flexible Unit Test Generation for Legacy HPC Code," *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.6

4. Melo, S. M., Souza, P. S. L., and Souza, S. R. S., “Towards an Empirical Study Design for Concurrent Software Testing,” *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.11
5. Smith, S., Jegatheesan, T., and Kelly, D., “Advantages, Disadvantages and Misunderstandings About Document Driven Design for Scientific Software,” *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.10
6. Pflüger, D., Mehl, M., Valentin, J., Lindner, F., Pfander, D., Wagner, S., Graziotin, D., and Wang, Y., “The Scalability-Efficiency/Maintainability-Portability Trade-Off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review,” *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.8
7. Pflüger, D. and Pfander, D. “Computational Efficiency vs. Maintainability and Portability. Experiences with the Sparse Grid Code SG++,” *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.7
8. Munipala, U., and Moore, S., “Code Complexity versus Performance for GPU-accelerated Scientific Applications,” *Proceedings of the Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, Nov. 14, 2016. Salt Lake City, USA. DOI:10.1109/SE-HPCCSE.2016.12
9. Carver, J. "First International Workshop on Software Engineering for Computational Science and Engineering." *Computing in Science & Engineering*. 11(2): 8-11. March/April 2009.
10. Carver, J. "Report from the Second International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE 09)." *Computing in Science & Engineering*. 11(6):14-19. Nov/Dec 2009.
11. Carver, J. "Software Engineering for Computational Science and Engineering." *Computing in Science & Engineering*. 14(2):8-11. March/Apr 2012
12. Carver, J. and Epperly, T. "Software Engineering for Computational Science and Engineering [Guest editors' introduction]." *Computing in Science and Engineering*. 16(3):6-9. May/June 2014.
13. Carver, J., Chue Hong, N., and Ciraci, S. "Software Engineering for CSE." *Scientific Programming*. Volume 2015. Article ID 591562. DOI: 10.1155/2015/591562
14. Carver, J. “Software Engineering for Science,” *Computing in Science & Engineering*, 18(2):4-5. March/April 2016.